International Journal of Electrical and Computing Engineering (IJECE)
Vol. 1, Issue. 4, June 2015 *ISSN (Online): 2349-8218*

Automated Detection of Root Cause and Fixing of Programs with Patches

T.Mamatha[1], Dr.D.S.R.Murthy[2], A.Balaram[3]
[1]Assistant Professor, Department of CSE, Sree Nidhi Institute of Science & Technology, Hyderabad, INDIA
[2]Professor and Head of CSE, Geethanjali College of Engineering and Technology, Hyderabad, INDIA
[3]Associate Professor, Department of CSE, CMR Institute of Technology, Hyderabad, INDIA

ABSTRACT: In automated debugging, first we reached a stage where fully automatic testing is happened with the help of so many automatic testing tools, than tried to find location of root cause, finally we are on search of automatic debugging. Automatic debugging is very essential for a developers, on this particular point of view so many researchers are done so much of work on this area, with their inspiration, We propose new idea for automatically fixing the bugs with the help of Cause Effect Graph, where it will found the root cause, after we found the root cause, we are applying the required patch for the given problem. For fixing the bug we are using fix analysis. Earlier so much work is happened on automatic debugging like genetic programming, bug fix…..all are of them has their own success rate in the same way, we propose new idea for automation debugging

Keywords: Cause Effect Graph, Decision Table, Fix Analysis

I. INTRODUCTION:

Automatic software repair is the process of giving solutions to a software bugs automatically. Without any human involvement an automatic software repair system fixes software bugs. The main purpose on doing this to save maintenance costs and to enable systems to be more resilient to bugs and unexpected situations. The fixing may happen at the level of the program code [16] (for instance by changing the conditional expression of an if statement) or at the level of the program state (for instance by changing the value of a register). An automatic software repair system focuses on one particular type of bug [16], in one particular technical space (e.g. initialization bugs in Java programs). It embeds algorithms and heuristics that leverage knowledge about this kind of bugs: on what the causes are, on how the symptoms manifest themselves, on what the likely fixes look like.

In this paper we propose new idea, after testing is finished, with the help of bug tracking tool we are going to give the details of bugs, like snapshots, steps to reproduce. how, when and at which place the symptoms are showed…..after that depending on the bug details given by the tester. With the help of cause effect graph and decision table root cause is detected and fixes the bug. For fixing the bugs, we used the fix analysis where the program code is executed, if it matches any of the causes the corresponding patch is going to kept.

Example : We consider the login page of any application, Login has the functionality that, After you give Valid Username, Password and after clicking ok button only it should go to home page otherwise it should stay in same page and give an error message that "Please enter Valid username or Password". If this functionality deviates than that could be detected and fixed with proper code patch. The sequence of flow is shown below.

Program module → testing → fault detected →bug tracking tool → cause effect graph → decision table → corresponding action as a patch

II. RELATED WORK:

Our idea was proposed based on previous works done by researchers like Jeffrey et al present bug fix, a tool that summarizes existing fixes in the form of association rules. Bug fix then tries to apply existing association rules to new bugs. User can also provide feedback in the form of new fixes or validation of fixes. Other authors use genetic algorithms to generate suitable fixes. Weimer, describe GenProg, a technique that Uses genetic programming to mutate a faulty program into one that passes all given test cases. Kim, describe Par, a technique that combines GenProg's Genetic programming with a rich predefined set of fix patterns (suggested by human written patches). Most of the fix patterns supported by Par are covered by AutoFix

synthesis. GenProg fixes 52% of 105 bugs with the latest improvement. Par fixes 23% of 119 bugs. Nauyen, build on previous work about detecting suspicious expressions to automatically synthesize possible replacements for each expression their SemFix Technique replaces or adds constant, variables and operators to faulty expression until all previously failing tests become passing.

Pachika, a tool that automatically builds finite-state behavioral models from a set of passing and failing test cases of a java program. Pachika also generates fix candidates by modifying the model of failing runs in a way which makes it compatible with the model of passing runs. The modifications can insert new transitions or delete existing transitions to change the behavior of the failing model. The changes in the model are then propagated back to the java implementation.

Auto fix exploits same of the technique used in pachika such as finite state models and state abstraction. It is based on contracts, which fixes 16 faults out of 42.

Some fixing technique works dynamically i.e at run time. Example data structure repair, Demsky and Rinard show how to dynamically repair data structure that violate their consistency constraints. Programmers specifies the constraints, which are monitored at runtime, the system reacts to violations of the constraints by running repair actions that try to restore the data structure in a consistent state.

III. CAUSE EFFECT GRAPH:

Here causes are the input conditions and effects are the results of those input conditions. Cause Effect Graph is a black box testing technique that graphically illustrates the relationship between a given outcome and all the factors that influence the outcome. It is also known as Ishikawa diagram as it was invented by Kaoru Ishikawa or fish bone diagram because of the way it looks.

A Boolean graph reflecting logical relationships between inputs (causes), and the outputs (effects) or transformations (effects).Cause-effect graphing describes a technique that uses the dependencies for identification of the test cases known as cause-effect graphing. The logical associations between the conditions (Causes) and their actions (effects) in a constituent or a system are represented is called as a cause-effect graph.

A.Cause and effect graph based on a situation:

Situation:

We consider the Login Page, where Login has some specifications

Username specifications:

1. Minimum 4 characters maximum 15 characters
2. First letter should start with alphabet followed by digits or alphabets
3. No special characters are used

Password Specifications:

1. Password should be at least 4 characters
2. Special characters and Digits to be used for strong password.

Depending on above conditions, outcome is, it should go to next page or Home page. Otherwise it should give invalid userid or password.

- The first character must be an alphabet. The second character may be a digit or alphabet
- If the username is incorrect (first character is not an alphabet or less than 4 or more than 15 characters), the message "Invalid Username" must be printed.
- If the password is incorrect (Less than 4 character), the message "Invalid Password" must be printed.

Solution:

Userid:

The causes for this situation are :
C1 – Minimum 4 characters Maximum 15 characters
C2 – First character is Alphabet followed by alphabet or digit
C3 – No special characters are used

Password:

C4 – At least 4 characters
C5 – Special characters and Digits are used for strong password.

The effects (results) for this situation are:
E1 – Go to next page or Home page
E2 – Print "Invalid userid"
E3 -- Print "Invalid Password"
E4 – Print "Try again"

First draw the causes and effects as shown below

Key – Always go from effect to cause (left to right). That means, to get effect "E" ,what causes should be true.

Effect E1: Effect E1 is to go Next Page or Home page

– Minimum 4 characters and Maximum 15 characters
– First Letter start with Alphabet followed by Letter or digit.
– No Special Characters are used.

Putting these 3 points in symbolic form:

For E1 to be true – following are the causes:
– C1, C2 and C3 should be true
– C4 and C5 should be true

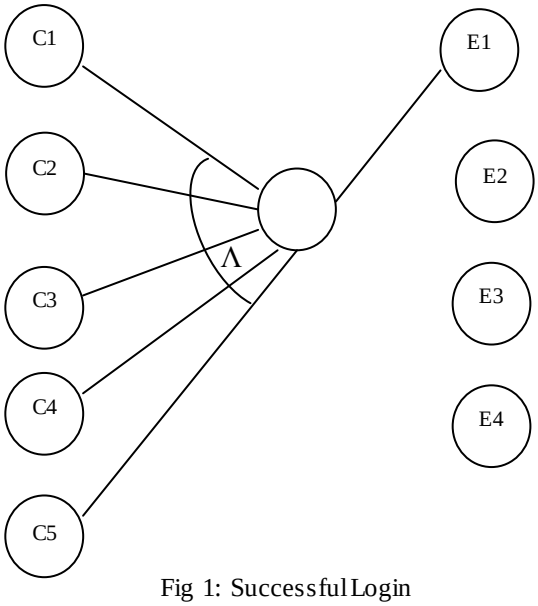

Fig 1: Successful Login

Effect E2 & E3: is to print "Invalid Userid and Password". When any of the condition is false.

Symbolic form:

For E2 & E3 to be True – following are the causes:
– If either of C1,C2 & C3 Fails
– If either C4 or C5 Fails

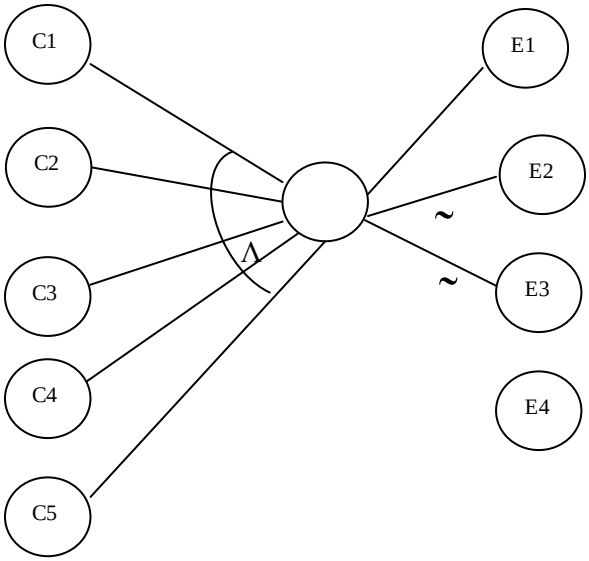

Fig 2: Unsuccessful Login

Effect E4: is to print "Try Again". When any of the condition is false

Symbolic form:
For E4 to be true – following are the causes:
– Either any of the C1,C2 or C3 Condition fails
– Either any of the C4 or C5 conditions Fails

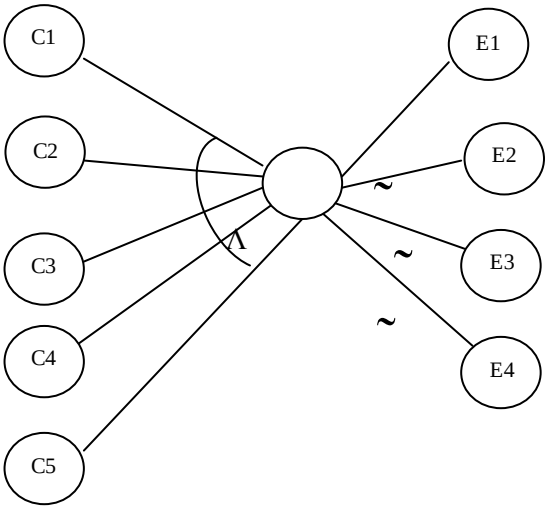

Fig 3:Trying Again on Unsuccessful Login

B. Decision table based on Cause Effect graph:

Conditions	Email Id	T	T	F	F	BLANK	BLANK	BLANK	VALID	INVALID
	Password	T	F	T	F	BLANK	VALID	INVALID	BLANK	BLANK
Actions	Expected results	T	F	F	F	F	F	F	F	F
	Show page	Home Page	Login page	Login page	Login page	Login page	Login page	Login page	Login page	Login page

Fig 4: Decision Table

A decision table is essentially a structured exercise to formulate requirements when dealing with complex business rules. Decision tables are used to model complicated logic. In a decision table, conditions are usually expressed as true (T) or false (F). Each column in the table corresponds to a rule in the business logic that describes the unique combination of circumstances that will result in the actions.

Here T means, it satisfies the Conditions and F means it will not satisfy the conditions. Decision table is shown in Fig(4)

C. Fix Analysis:
In Fix analysis we used a set of predefined templates called *fix patches*. After finding the root cause, the corresponding Fix patch is going to be placed in the fault code.

Sample code for validating Login :

```
function validate(){
 if(reg.Name.value == "" || reg.Password.value == ""){
   alert("please enter all details");
  }
else{
   nm = reg.Name.value;
   len = nm.length;
   if(len>16 || len<4)
   {
     alert("name must contain atleast 4 characters and should less than 16 characters");
   }
   pw = reg.Password.value;
          len = pw.length;
   if(len>16 || len<4)
  {
     alert("Password must contain atleast 4 characters and should less than 16 characters");
   }
          if(len<16 && len>=4)
          {
 var pass_match = /([@]|[#]|[$])/;
 if(pw.match(pass_match)==null)
 {
  alert("Password must contain special characters to be strong ");
 }
else
 {
  alert("Password is strong ");
 }
```

For the above code, if the login is allowing the values below 4 characters and that if found in testing and reported than with the help of cause effect graph root cause is found corresponding Buggy code is replaced with Patch code.

Buggy code:

```
if(len>16 || len<3)
   {
     alert("Password must contain atleast 4 characters and should less than 16 characters");
   }
```

Replaced with Correct patch:

```
if(len>16 && len<4)
   {
     alert("Password must contain atleast 4 characters and should less than 16 characters");
   }
```

IV. Conclusion:

The approach described in the paper, is an important contribution towards the ideal of

automatic debugging. Cause-Effect graph helped so much in finding the root cause so that fixing the bug at correct place is happened easily and it very important in automatic debugging process.

V.References:

[1] Y. Wei, Y. Pei, C. A. Furia, L. S. Silva, S. Bucholz, B. Meyer, and A. Zeller, "Automated fixing of programs with contracts," in ISSTA, 2010.

[2] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in ICSE, 2012.

[3] Claire Le Goues, Stephanie Forrest, and Westley Weimer. Current challenges in automatic software repair.Software Quality Journal, 21:421{443, 2013.

[4] R. Abraham and M. Erwig, "*Test-driven goal-directed debugging in spreadsheets*", *IEEE Symposium On Visual Languages and Human Centric Computing*, pp. 131–138, 2008.

[5]Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. SemFix: program repair via semantic analysis. In Proceedings of the 35th International Conference on Software Engineering, pages 772–781.IEEE Press, 2013.

[6] Fernando Netto, Marcio Barro, Adriana C.F. Alvin, An Automated Approach for Scheduling Bug fix Tasks-2010 Brazilian Symposium on Software Engineering

[7]A. Arcuri and X. Yao, "*A novel co-evolutionary approach to automatic software bug fixing*", IEEE Congress on Evolutionary Computation, pp. 162–168, 2008.

[8]Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, Westley Weimer. *GenProg: A Generic Method for Automatic Software Repair*. In « IEEE Trans. Software Eng. », 1, 38, 2012, 54-72.

[9] Martin Monperrus. *A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair*. In « Proceedings of the International Conference on Software Engineering », 234-242, 2014,

[10] Demsky B and Rinard, M, "*Automatic detection and repair of errors in data structures*", *ACM SIGPLAN Notices, 38*(11), 78, 2003.

[11] W. Weimer, T. Nguyen, C. Le Goues, and S.Forrest, "*Automatically finding patches using Genetic programming*", ICSE, 2009.

[12] R. Abraham and M. Erwig, "*Goal-directed debugging ofspreadsheets*", *IEEE Symposium on Visual Languages and Human-Centri Computing*, pp. 37–44, 2005.

[13] T. Ackling, B. Alexander, and I. Grunert, "Evolving patches for software repair," in Genetic and Evolutionary Computation,2011, pp. 1427–1434.

[14] C. Weiß, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in Workshop on Mining Software Repositories, May 2007.

[15] D. Jeffrey, M. Feng, N. Gupta, and R. Gupta, "BugFix: A learning-based tool to assist developers in fixing bugs," in International Conference on Program Comprehension, 2009.

[16] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "DebugAdvisor: a recommender system for debugging," in Foundations of Software Engineering, 2009, pp.373–382.

AUTHORS:

Mrs. T.Mamatha, Working as an Assistant Professor in Dept of C.S.E of SreeNidhi Institute of Science & Technology. M.Tech in Software Engineering from JNTU, Anantapur. B.Tech in C.S.E from JNTUH. Her Area of Interest is Software Engineering & Software Testing.

Dr. D. S. R. Murthy is currently working as a Professor & Head of the Dept of CSE in Geethanjali college of Engineering and Technology since june 2014.Earlier he worked as a professor of Information Technology in Sreenidhi Institute of science and Technology from OCT 2004 to JUNE 2011. Prior to that he was in NIT Warangal JNTUCE, Anantapur and ICFAI in different faculty positions of Computer Science. He is a Fellow of IETE, Senior Life Engineer (EI(I) & IETE). He published a text book on C Programming & Data Structures. His research interests are Image Processing, Image Cryptography and Data Mining.

Mr. A.BalaRam working as Associate Professor in the Dept. of Computer Science and Engineering, CMR Institute of Technology, Hyderabad, India. He has more than 10 years of teaching experience. . His research interests are Software Engineering, Image Processing, Net work Security and Cryptography.